Architectural Challenges and Solutions for Petascale Postprocessing*

Hank Childs†
Lawrence Livermore National Laboratory

Abstract

THIS IS A DRAFT!!

With petascale applications comes petascale data; gleaning knowledge and insight from large-scale data is widely accepted to be a limiting factor in many fields of scientific endeavor. Large-scale data presents two incredible challenges: scale and complexity. The scale challenge refers to the problem of being able to process tremendous numbers of bytes of data within some time constraint. The complexity challenge is more subtle; petascale data is inherently complex and reducing this data to comprehensible forms is difficult.

For the scale challenge, it is often assumed that continued usage of viable techniques for the terascale will provide a recipe for success at the petascale. We argue the opposite; following this path will result in cost prohibitive solutions. Instead, we must pursue "smarter" techniques, such as in situ and multi-resolution processing, which are well established by a decade's worth of research.

The forms of analyzing data are very diverse, but they can be summarized in broad use cases: data exploration, presentation graphics, quantitative analysis, comparative analysis, and visual debugging. Together, these use cases are responsive to the complexity challenge. Moreover, these use cases are responsive to the demands of the user community. Making pretty pictures is just one component of the simulation community's postprocessing needs and all of these needs must be addressed for petascale data.

A major focus of this paper is to construct a road map for realizing this broad set of use cases at the petascale, and, further, to do this in the most economical way possible. None of the smart processing techniques we consider are a panacea; none are capable of supporting every use case. However, a software architecture that underpins these processing techniques and that can dynamically select between them will provide an economical way to meet these challenging requirements, and in this paper we discuss such an architecture.

*This is UCRL-TR-232039.
†e-mail: childs3@llnl.gov

1 Introduction

As noted in the abstract, gleaning knowledge and insight from large-scale data is widely accepted to be a limiting factor in many fields of scientific endeavor. As the computational science community achieves petascale level simulations, many challenges will be created or will worsen in the area of postprocessing. In this paper, we consider a system (that is, a software architecture) to enable petascale postprocessing in many diverse forms. The importance of such a system is high. Continuing on the current trajectory for terascale postprocessing will result in cost prohibitive hardware purchases. Further, more and more participants are entering the supercomputing arena. By delivering capability in the form of a system, this investment will be usable by many and significantly more cost effective than the alternative: implementing many tailored solutions, with much redundant capability, for each participant.

In section 2, we describe "pure parallelism", the technique commonly used for production visualization and analysis at the terascale. We also describe why this technique will likely be cost prohibitive at the petascale. In section 3, we survey the "smart" alternatives to pure parallelism and discuss their relative costs. In section 4, we argue that many use cases that go beyond traditional visualization should be considered for two reasons. One, this is responsive to the mandate of the user community. Techniques are needed to reduce the incredible complexity of petascale data and these diverse use cases will enable that. Two, the increased cost of entry for petascale postprocessing will result in economies from considering these use cases simultaneously. In section 5, we survey the use cases themselves: data exploration, quantitative analysis, presentation graphics, visual debugging, and comparative analysis. In section 6, we discuss the viability of our smart processing techniques for each of the use cases, and argue that a combination of techniques is needed to provide the most cost effective solution. Finally, in section 7, we describe a vision for a system that allows for algorithms to be implemented in a processing technique "indifferent" manner.

2 The problem with pure parallelism

"Pure parallelism" refers to the technique where many processors are used to read the entire data set into primary memory. This is the simplest technique to implement; the data is stored in its natural form and every byte of data is available on demand. At the terascale, production visualization tools, such as VisIt[4], EnSight[5], and ParaView[9], successfully use this technique for the processing of data. They partitioned the data and had each processor operate on its respective subset, with each tool allowing varying levels of communication amongst its processors during execution.

The hardware to enable this processing takes one of two forms. With one form, a mini-supercomputer, dedicated to visualization and analysis, is used offline. In this case, the data is either transferred to a private disk for the machine, or the data is directly accessible on the supercomputer through a shared disk. With the other form, the supercomputer itself is used to visualize and analyze the results after the simulation has completed, either through dedicated postprocessing nodes or by using the same batch nodes used to run the simulation.

The problem with continuing on the pure parallelism path is that the hardware costs will be prohibitive. Lawrence Livermore's 360 TeraFlop BlueGene/L machine has a dedicated visualization and analysis machine named Gauss, which is a 512 processor Linux cluster. Gauss itself made the Top500, and cost between one and two million dollars. When planning for a similar setup for a five PetaFlop machine, the estimate for the visualization and analysis cluster was *fifteen million dollars*. Although this cost *may* be justifiable in the context of the larger supercomputer purchase, it is not justifiable given that software alternatives can reduce the hardware requirements.

The notion of co-opting a portion of the supercomputer to do interactive postprocessing is also not very palatable. Postprocessing is often "bursty", meaning that processing is done and then a user sits and looks at the results for a period of time. Then more results are requested, and so on. Is this portion of the supercomputer to sit idle (during a "non-burst"), waiting to respond? Is that cost justifiable? If the supercomputer does not sit idle, then data would have to be repeatedly sent to disk and later brought back in to primary memory. Will the ensuing thrashing completely remove any chance at interactive results?

Another problem with co-opting the supercomputer is the trend toward multi-core processing. Postprocessing requires a different portfolio of computing resources than simula-

tion; it is typically a memory- and I/O-limited activity and is rarely compute-limited. As petascale platforms tend towards increasing compute power, with memory and I/O not keeping pace, the net increase in hardware capability delivered for postprocessing is not proportionate. Hence, it will need greater resources to handle the considerably larger data sets, and the postprocessor will need an increasing proportion of the supercomputer to do its job.

To summarize, planning on using pure parallelism at the petascale either means making an expensive computer purchase, or it means that the supercomputer's effectiveness is going to be substantially diminished.

3 Large Data Processing Techniques

In this section, we survey the large data processing techniques that have already been investigated and/or developed by the community. Many of these techniques perform well with certain postprocessing use cases, but not others. In section 5, we enumerate these use cases. In section 6, we describe matches between use cases and techniques.

For each section, we start by describing the technique, then describe its advantages, disadvantages, and challenges. Disadvantages are inherent problems with the technique; challenges are things that are not currently done well with the technique, but could be improved.

3.1 In-situ processing

3.1.1 Description

In-situ processing consists of processing the data using the resources allocated for the simulation code. In this model, the simulation code advances in time for a while, then hands off a baton to a postprocessing algorithm, which generates results and hands the baton back[1].

3.1.2 Advantages

This solution bypasses I/O, which is a major advantage, and it also ensures that sufficient compute resources are available.

[1]Note that this model blurs the line of the term "postprocessing", since analysis is done while the simulation code is running, rather than after it has finished. Regardless, we continue to use the term, because the analysis is still performed after the simulation code has calculated a result, even though the period of time afterwords is only fractional seconds.

3.1.3 Disadvantages

In situ processing requires that the user knows exactly what analysis should be done as the simulation is running, which is often not the case. Further, there is an entire class of operations that look at an event and want to trace *backwards in time* how that event came to be. This is not possible with in-situ processing.

Finally, there is a tendency to bypass I/O altogether when in situ processing is employed. Once a simulation is finished, the only results that can be analyzed are the outputs created during the in situ processing. If these results are perplexing, there is no way to perform additional analysis aside from running the simulation again.

3.1.4 Challenges

A major challenge with in-situ processing is developing a single solution that will be effective for many simulation codes. Postprocessing algorithms often fix the way it represents different mesh types and fields on those meshes. For example, a postprocessing algorithm may assume the fields are in "row-major order". But each simulation code has its own in-core representation. So one may have "row-major order", but another may have "column-major order". In this case, the postprocessing algorithm can operate on the "row-major order" simulation directly, but must convert the data from the "column-major order" simulation so it can effectively operate on it. This conversion step leads to memory bloat and is thus disastrous; high end supercomputers are short on memory and thus the simulations that run on it are often memory bound. Tolerating this conversion step will effectively place additional limitations on the resolution these simulations can obtain. (It should be emphasized that the row-major / column-major example is just one of *many, many* possible pitfalls. Experience shows that the odds of a fixed data model of a postprocessing tool matching that of a simulation code are very low, unless the postprocessing tool was built around the simulation code.)

The challenge, then, is to implement algorithms in a way that is indifferent to the simulation code's in-core layout, and, of course, still maintain high efficiency. One approach is to leave the in-core layout as is and use abstract interfaces (e.g. virtual functions) to hide the data layout from the postprocessing algorithms. Depending on the extent that virtual functions are used, this may be sufficiently efficient. How-

ever, the SCIRun library[8] demonstrated an even more successful approach to this problem, by using a templated data model. This templated approach allows for high efficiency, in addition to solving the data layout issue.

Finally, performing in situ processing on petascale computers would force common postprocessing algorithms to be run on an unprecedented numbers of processors. The extent to which these algorithms will be efficient at this scale is an open question.

3.2 Multi-resolution techniques

3.2.1 Description

With multi-resolution techniques, the data set can be explored by starting with a coarse representation and then focusing in on portions of the data set to see finer representations. Finer representations may come automatically after coarse representations finish calculating, or they may be explicitly requested by the analyst. Ultimately, enough replacements with finer and finer representations will result in seeing the original data (at least desirably).

3.2.2 Advantages

These techniques have been greatly studied by the research community [add citations], so their viability is well established. For some use cases, they are tremendous wins, since they alleviate I/O and processing costs.

3.2.3 Disadvantages

For other use cases, offering results on a coarse data set is not meaningful, because too much detail is lost. Further, the simulation code's outputting of the data set may now involve additional computation and potentially additional bytes to create the multi-resolution representation. Given that most simulations output at a low rate (to save on slow I/O), this overhead may be low.

3.2.4 Challenges

When a coarse version of the data set is displayed, it is important to understand the accuracy of the results. Some multi-resolution techniques emphasize performance through elegant indexing, but de-emphasize control over what data is displayed at coarse resolutions and de-emphasize understanding of the error of the intermediate results.

Finally, some multi-resolution techniques lose their knowledge of the original data set. Some use cases require that results be returned in a format familiar to users. For example, a query that returns an element identifier for a block-structured simulation would want that identifier to include information like which block the element is contained in, as well as its logical I, J, and K indices. The challenge is to apply a multi-resolution hierarchy to this data with minimal overhead, and still be able to produce such information.

3.3 Out-of-core

3.3.1 Description

"Out-of-core" techniques[14] consist of partitioning the data set into subsets, and then processing each subset one at a time. It is assumed that each subset is small enough to fit into primary memory. After processing each subset, it is discarded, in all or in part, before moving on to the next one.

3.3.2 Advantages

This technique provides a cost effective alternative to pure parallelism. By reducing the memory needs, it can process any amount of data.

3.3.3 Disadvantages

This technique will not satisfy use cases with interactivity requirements - the processing time will often be prohibitive. This the technique can be parallelized, but we do not consider parallelized out-of-core in this paper, as its purpose is to increase interactivity and the number of processors to achieve this interactivity will approach that of pure parallelism.

3.3.4 Challenges

Some algorithms require having all of the data in memory (for example to perform collective communication). These algorithms are, at a minimum, difficult in this setting, because data must be read multiple times. However, clever implementations of these algorithms may be able to minimize re-reads, or even eliminate the re-reads altogether in some cases.

3.4 Pure parallelism

3.4.1 Description

"Pure parallelism" uses many processors to read the entire data set into primary memory. The data is stored in its natural form and every byte of data is available on demand. We argued in section 2 that this technique will be cost prohibitive. Despite this, this technique will likely have a role in the petascale future (primarily as a backup). The question is to what extent.

3.4.2 Advantages

This processing paradigm is suitable for all use cases we consider.

3.4.3 Disadvantages

The hardware costs for pure parallelism will be large.

3.4.4 Challenges

I/O represented the significant bottleneck at the terascale, and, given the respective rates that processing power and I/O are advancing at, this problem will only worsen in the petascale regime. Further, many of the future petascale supercomputers being considered have lightweight operating systems that create barriers for deploying tools, for example precluding sockets, shared libraries, virtual memory, and/or threads. Both of these issues must be addressed for a pure parallelism approach to be successful. (This latter issue also exists for in situ processing.)

3.5 Cost comparison of techniques

Calculating exact costs for these techniques is difficult and highly dependent on the issue being solved. However, the differences in cost between the techniques are large enough that they can be assessed in gross terms. The two axes of comparison to consider are machine costs and runtime costs.

Machine costs measure the cost to perform the processing. For pure parallelism, this is the cost of procuring and maintaining a postprocessing-oriented supercomputer or the cost of dedicating a portion of the supercomputer for these purposes. For in situ processing, the cost is the cost of having the supercomputer do the postprocessing work and *not* be advancing in the simulation. However, the costs are substantially reduced because the data is already loaded in main memory, saving the very expensive I/O costs. For multi-resolution techniques, the majority of the cost comes from the overhead mentioned in section 3.2.3 of outputting the multi-resolution information. The rest of the processing is then done with a desktop computer or a small cluster. Out-of-core techniques produce the lowest costs, as they do not place overhead on the simulation code and also use a desktop computer or a small cluster.

Runtime costs measure the cost to a user to do the required analysis. For pure parallelism, this is the cost of the parallelized I/O and processing. Given adequate resources, this can be very fast. For in situ processing, the cost is even less, since the adequate resources are guaranteed and I/O is not a factor. For multi-resolution techniques, the cost is also fairly cheap (provided that multi-resolution is a general match to the use case). Out-of-core processing is far and away the most expensive, to be the point of being prohibitive in some cases.

Table 1 summarizes the preceding paragraphs. For runtime costs, multi-resolution techniques were listed as cheaper than pure parallelism because they often can avoid severe I/O costs. (It could be reasonably argued that these two should be reversed.)

Technique	Machine Cost (lowest is best)	Runtime Cost (lowest is best)
Pure Parallelism	4	3
In Situ	3	1
Multiresolution	2	2
Out-of-core	1	4

Table 1: The machine and runtime costs of the processing techniques.

It is difficult to combine our non-quantitative runtime and machine cost tables into a single cost. That said, we declare in situ and multi-resolution techniques to be "better" than out-of-core and pure parallelism techniques, as the runtime costs for out-of-core are so high, as are the machine costs for pure parallelism. This gives a "total cost" table (table 2).

Technique	Cost (lowest is best)
Pure Parallelism	2 (tie)
In Situ	1 (tie)
Multiresolution	1 (tie)
Out-of-core	2 (tie)

Table 2: The "total" costs of the processing techniques.

4 THE ARGUMENT FOR DIVERSE USE CASES

A principal assumption of this paper is that it behooves both developers and users to consider a diverse set of use cases. In this section, we provide two arguments in support of this assumption. One, the user community relies on diverse use cases that go beyond classic scientific visualization and expect that these approaches will be available at the petascale. Further, petascale data is inherently complex and a diverse suite of techniques are needed to reach crucial insights. Two, an economy is achieved when the diverse use cases are addressed simultaneously.

4.1 Beyond Meat Grinders

"Meat grinder" is a term used to describe tools that focus solely on handling the scale of data without considering the associated complexity. The analogy is that these tools simply change the form of the data (from meshes to images), cranking data through without bothering to make it easier to interpret. Complex isosurfaces often go beyond what our human brain's visual processing system can meaningfully understand. Following the analogy, then, we should spend more time "finding the right meat." To truly understand petascale data, addressing the scale is necessary, but not sufficient. Instead, we need to complement techniques such as isosurfacing, slicing, and volume rendering with techniques that more directly allow analysts to do the science they need to do. This often comes in diverse forms, such as:

- quantitative measures, like calculating a probability density function,
- comparative techniques, like studying why a high resolution simulation doesn't match smaller predecessors,
- the ability to dig down and study the behavior of just a handful of data points.

Focusing exclusively on data exploration will deny analysts what they need: a diverse suite of techniques to reach crucial insights.

4.2 Economies Achieved by Considering Diverse Use Cases

To what extent should there be a common infrastructure for delivering these diverse use cases? At one extreme, does it make sense to target a single tool/library that can employ each of the postprocessing techniques from section 3 for our target use cases? Or are there divisions that make multiple tools/libraries useful and/or necessary? We believe the answer is that combining the solutions for these postprocessing areas into a single solution is beneficial from a cost perspective. We will make this argument in the following sections, from both a development and a user perspective.

4.2.1 Development Economies

The techniques described in section 3 all have higher software costs to implement than that of pure parallelism. The cost of entry for postprocessing will rise for the petascale regime. Further, many distinct use cases can be solved with the same or similar solutions. Hence, from a development perspective, there are economies that make a single tool/library (or small set of tools/libraries) desirable. The biggest example is that a large data infrastructure can be developed once and reused in all of these areas (the nature of this infrastructure will be discussed in section 7). In addition, file format readers, algorithms, and data models all benefit from this reuse. Finally, many solutions in this space previously (for example, AVS[16], VTK[11], and OpenDX[1]) have shown that a flexible, extensible architecture is advantageous, and that would also be reusable. The advantage, then, of a single tool/library is that all of these assets can be developed a single time and reused. Finally, the system we propose in section 7 will allow algorithms to be implemented in a way that is indifferent to processing technique. So a developer can implement an algorithm one time, and achieve a greater economy; the algorithm would then be deployable with pure parallelism, out-of-core, in situ, and multi-resolution processing[2].

4.2.2 User Economies

From a user perspective, there are economies to having a single tool (*not library*) that can perform all of these use cases. First, visualization and analysis tools tend to have complex features, and hence complex interfaces. Learning one interface, as opposed to five or more, is often preferred by users. Further, users tend to jump back and forth between use cases. For example, a session focusing on comparative analysis often leads to some key discovery that then leads to visual de-

[2]Some algorithms place constraints on the types of processing they allow; this issue will be discussed further in section 7.

bugging on one of the simulations. From within a single tool, they can do this seamlessly. With multiple tools, there is overhead.

4.2.3 Realities of General Solutions

The argument is that, especially for petascale postprocessing, there is an advantage to having a system (software infrastructure) where assets can be re-used, and, for users, there is an advantage to having a single interface to perform all of their use cases. This should be tempered by experience, however. In the world of visualization tools, there is no "one size fits all" solution. Some scientific domains are so specialized that there is little benefit from leveraging a generalized postprocessing infrastructure. Conversely, making a postprocessing infrastructure that is so general that it can support all scientific domains can substantially increase development time and reduce performance.

5 Postprocessing Use Cases

In section 4, we motivated the power of a diverse set of use cases. They are what our customers demand, they will allow us to better understand complex data, and we can achieve economies by considering them simultaneously. In this section, we explore each of the use cases: data exploration, quantitative analysis, comparative analysis, visual debugging, and presentation graphics. For each use case we will present a description, discuss requirements placed on processing techniques, and (for only some of the use cases) discuss new challenges created by large scale data.

5.1 Data Exploration

5.1.1 Description

Data exploration includes the standard scientific visualization techniques, for example slicing, contouring, and volume rendering. These techniques employ our sophisticated visual processing system to quickly assimilate the gross trends of a simulation and also pick out irregularities.

5.1.2 Requirements for Processing Techniques

With respect to processing, we need to keep the "bursty" model of data exploration in mind. It is expected that data exploration is interactive. Further, it must be understood that data exploration has periods of interactivity, when results are considered, and that idle time must be weighed against the resources used to produce results.

5.1.3 Large Data Challenges

There are only so many pixels on a computer screen (and only so many rods in your eyes). Petascale simulations have so much data that many, many data points will be encoded in a single pixel. We need to ask to what extent classic data exploration techniques must be adapted to guarantee that what is presented to an analyst accurately reflects the simulation.

5.2 Quantitative Analysis

5.2.1 Description

"Quantitative analysis" is a broad term and it is useful to think of three areas:

- Basic techniques that span scientific domains
- Advanced techniques that span scientific domains
- Techniques that are tailored to a specific physics area.

The motivation for artificially distinguishing between basic and advanced techniques is that some people hear the phrase "quantitative analysis" and think of *only* basic operations such as integrating density over a volume to get a mass, calculating a surface area, or calculating a flux. These operations are often among the most useful, but, to many others, this represents only the beginning. Examples of advanced techniques are calculating characteristic functions, such as chord length distributions[10], or considering regions of the mesh as a graph and identifying the connected components [add citation]. Techniques tailored to specific physics areas vary highly, but a good example is that of "virtual diagnostics," which calculate what a simulation would produce to some virtual diagnostic, allowing for a simulation to be compared to an actual experiment (through comparison to its real diagnostic).

5.2.2 Requirements for Processing Techniques

Techniques tailored to a specific physics area create the most new requirements. Some of these techniques may require incorporating outside knowledge, such as equation of state information. Further, many of these techniques require development from personnel that have, for example, a background in physics. So the development environment must

require that people with a different set of skills (i.e. non-visualization personnel) be able to effectively interact with the environment provided to them. (This environment may have many instantiations. One example is a stripped down programming environment, where personnel with domain expertise can implement algorithms in the C programming language on arrays of data. Another example could be a scripting environment where experts use a language, like Python.)

Finally, these techniques must often be performed at the full resolution of the data, not on simplified versions.

5.2.3 Large Data Challenges

It should be noted that, even at the terascale, this final area was often addressed by serial tools written directly by the simulation code developers and their customers. Anecdotally, these serial tools will run for a month to get an answer. A similar approach at the petascale, assuming a one thousand fold increase in data size, would lead to one thousand months – over eighty years! Where one month may be an acceptable turnaround time, eighty years definitely will not be. So the standard operating procedure at the terascale will lead to poor results at the petascale. Of course, these serial, specialized analysis tools could be parallelized, but, again, petascale has raised the cost of entry. The point, of course, is that petascale simulations are creating a need for a customizable petascale analysis environment.

5.3 Comparative Analysis

5.3.1 Description

Comparative analysis is also a broad term. Shen et al.[12] define three basic areas of comparative visualization: image-based comparison, data-level comparison, and topological comparisons. Image-based methods compare images that result from visualization algorithms. Most image-based comparisons[6, 17, 18] perform image differencing algorithms on images from multiple inputs. These systems are limited to comparing visualizations where the entire data set can be represented by a single image. These techniques are extremely important in the context of comparison to experimental results, such as is the case with [6] and [18]. Data-level comparisons place fields from one mesh onto another mesh and use derived quantities to examine differences between the two data sets. This technique has been employed in varying forms, such as in [12, 13, 15]. Topology-based (or feature-based) techniques compare features of the data sets [2, 7]. Typically, they survey the data set, create summaries of the data, and develop methods to visualize and compare these summaries.

5.3.2 Requirements for Processing Techniques

This requires processing multiple simulations simultaneously or multiple time slices simultaneously. Finally, these techniques must also often be performed at the full resolution of the data, not on simplified versions.

5.4 Visual Debugging

5.4.1 Description

For simulation code developers, visual debugging is often the number one use case for a postprocessing tool. This set of techniques are used for finding bugs in the simulation code. Examples of these techniques include locating a hot spot, finding an area of mesh tangling, or "picking" on an element and getting a report about it, such as the value of the element, the value of its neighbors, etc.

From the perspective of data processing, there are two major classes of visual debugging: aggregate queries and individual queries. An example of an individual query is a code developer noticing a hot spot and wanting to obtain more information about it. An example of an aggregate query is a code developer wanting to survey the entire data set and obtain a list of elements that meet some criteria. The difference between these two classes is the resources necessary to satisfy these queries.

5.4.2 Requirements for Processing Techniques

This use case creates an often overlooked requirement for the postprocessing tool: that is be able to produce information about the simulation in a manner that respects the original layout of the simulation. For example, any query that returns an element identifier must return the information in a way that makes sense to the simulation code developer.

5.5 Presentation Graphics

5.5.1 Description

Presentation graphics is one of the most time consuming activities for visualization experts. This term includes

moviemaking and making images for viewgraph presentations, publications, etc. Again, we distinguish between two classes of presentation graphics: "presentation-oriented graphics" and "exploration-oriented movies." "Presentation-oriented graphics" are typically high quality. Their purpose is to inform a large audience. "Exploration-oriented movies" are typically done by analysts for themselves. They make simple movies so they can observe, and hopefully ultimately understand, phenomena in the simulation. In many ways, this use case is very similar to data exploration, with the key differences being that the explorations are saved in a permanent record (for example, an MPEG). Both generally display images and use techniques that are similar to data exploration and both require various forms of automation of algorithms (i.e. animating a slice, an isovalue, or over time). Presentation-oriented graphics also augment the images with annotations like time sliders, logos, and text.

5.5.2 Requirements for Processing Techniques

With presentation-oriented graphics, it is assumed that the images represent the entire data set, as the images (even if they are frames of a movie) are often studied in great detail and there is often a desire to "show off" the full resolution of the data. This requirement is relaxed with exploration-oriented moviemaking; the movies often don't require the full resolution of the data.

6 Viability of Techniques for Use Cases

Having surveyed the processing techniques (section 3) and the use cases (section 5), we can see that not all of the smart processing techniques will be applicable to the use cases. The mode of failure always follows the same formula: processing technique "ABC" is unable to do "XYZ", which is required by use case "123". In section 6.1, we will enumerate the failure modes. This is followed by a summary of the viability of techniques for the use cases, in section 6.2, which provides a road map for the petascale postprocessing of our uses cases.

6.1 Failure modes

The reasons that a technique cannot work with a use case – the "XYZ"s – fall into one of four categories:

- Processing the data in its full and native form
- Supporting multiple data sets simultaneously
- Interactivity
- Idle time

6.1.1 Processing the data in its full and native form

Multi-resolution techniques do not process the data in its full and native form[3]. This is a requirement for all areas of quantitative analysis, comparative analysis, aggregate queries for visual debugging, and presentation-oriented graphics. Therefore none of those use cases can be fully solved with multi-resolution techniques.

It should be noted that multi-resolution may still have value added in the areas of comparative analysis and quantitative analysis. In this case, "rough" answers can be obtained, which can then be used to forge more in depth queries in the data set. At some point, however, both comparative analysis and quantitative analysis almost always dictate getting the results at the true resolution of the data. (Note that the system described in section 7 will allow for these results to be taken on a coarse representation, putting the decision in the hands of the user.)

6.1.2 Supporting multiple data sets simultaneously

In situ processing will only work with the simulation's current data set. So it is not able to support multiple data sets simultaneously. Further, comparative analysis requires processing multiple data sets (e.g. multiple time slices from one simulation, multiple simulations, etc.). It should be noted, however, that clever configurations could allow for a match here, for example having a client connecting to two in situ-based servers simultaneously. Regardless, on the whole, in situ processing and comparative analysis are not congruent.

Some advanced and physics-based quantitative analysis techniques have a time component, which can be difficult (going forward in time) or impossible (going backwards). So in situ processing should be considered on a case-by-case basis for these areas.

[3] They can do this with enough refinement, but this essentially becomes out-of-core in this case.

6.1.3 Interactivity

Out-of-core processing is not able to provide interactivity[4]. Further, data exploration and aggregate queries from visual debugging require interactivity. So out-of-core is not appropriate for these use cases. (Individual queries from visual debugging also require interactivity, but these techniques do not require petascale data processing, making out-of-core a viable approach.)

6.1.4 Idle time

In situ processing should not be used when there will be a substantial amount of idle time between processing requests. That is not a good use of the supercomputer. However, data exploration and visual debugging fit this model, so, in general, in situ processing should not be used with data exploration.

Of course, in some cases, the ability to ask questions interactively can be a time saver, for example when the alternative is to save the state of the simulation, analyze it offline and then later reload it. So this is capability that should ideally be available, but we should not plan on meeting "interactive probing" use cases through in situ processing.

6.2 Summarizing the viability of processing techniques for use cases

The last section established which processing techniques were suitable for our use cases. We summarize these findings in table 3. We incorporated the cost table (Table 2) to decide which processing techniques are preferred for addressing each of our use cases. We color the preferred techniques green. In the case of in situ processing, we also choose a backup solution, colored yellow, since in situ processing can only be used when the user knows what analysis should be done a priori.

Table 2 declared a tie between in situ processing and multiresolution techniques. From table 3, we see that "resolving the tie" between these techniques is not necessary as they are mutually exclusive for the use cases we consider. Further, in situ processing or multi-resolution techniques can be used to address many of our use cases, and only comparative analysis falls back to a "high cost" solution. Since each row has at least one "Yes", the set of use cases we want to consider is fully covered by our processing techniques; these techniques should be sufficient to do petascale processing. In the cases where in situ was selected, either out-of-core processing or pure parallelism play a larger role as a backup. Choosing between out-of-core and pure parallelism force a trade off to be made between runtime costs and machines costs, respectively, which should be taken on a case-by-case basis.

Finally, although the presentation of this information is somewhat black-and-white (Yes or No), there are exceptions to some of these cases, especially when a group is listed as "No."

7 An Architecture for Supporting Petascale Postprocessing Use Cases

Section 4 established the motivation for considering many use cases simultaneously. Section 6.1 showed us that none of our smart alternatives could single-handedly provide a way to process petascale data. Section 6.2, however, provided a road map for employing a combination of these smart techniques based on the type of work being done. In this section, we describe a system that will allow us to utilize this road map, to deliver all of our use cases at the petascale, and in an economical way.

The vision is to have a single system (or software architecture) that can facilitate the processing of petascale data. In this proposed system, algorithms can be implemented in a manner that is indifferent to the processing paradigm, at least indifferent to the extent possible. This will be done by using data flow networks (described in section 7.1) and then controlling how data flows through these networks (described in section 7.2).

7.1 Data flow network overview

The most prevalent design for postprocessing libraries is data flow networks, for good reason. In this system, algorithms are implemented in a manner that focuses only on its input and output. This allows for algorithms to be interoperable, to be "plugged together" in dynamic ways by users to fit their current needs. Further, it is easily extensible; a new algorithm can be implemented by simply creating a new module.

Data flow networks are not without faults. They normally require the data model to be somewhat fixed[5] and each al-

[4] Recall that we are considering serial out-of-core processing, since parallelization to obtain interactivity will approach pure parallelism.

[5] Fixed in the sense that adding new fields type (like edge-centered data) or new meshes (like ying-yang meshes) is difficult.

	Pure Parallelism	In Situ	Multires	Out-of-core
Data Exploration	Yes	No	Yes	No
Quantitative Analysis				
Basic	Yes	Yes	No	Yes
Advanced	Yes	Sometimes	No	Yes
Physics-based	Yes	Sometimes	No	Yes
Comparative Analysis				
Image-Based	Yes	Yes	No	Yes
Data-Level	Yes	No	No	Yes
Topological	Yes	Sometimes	No	Yes
Visual Debugging				
Individual Queries	Yes	No	Yes	Yes
Aggregate Queries	Yes	Yes	No	?
Moviemaking				
Presentation-oriented	Yes	Yes	No	Yes
Exploration-oriented	Yes	Yes	Yes	Yes

Table 3: The viability of postprocessing techniques for each of the use cases. If a technique is viable for a use case, it is labeled "Yes", "No" otherwise. For each use case, a preferred solution is selected and colored green. Back up solutions are colored yellow.

gorithm must be aware of the data model, or, at a minimum, state which parts of the data model they support. Further, the modular design means that algorithms are executed one at a time, rather than all at one time. The latter is typically more efficient (for one, it avoids thrashing memory when the data being processed at that moment is sufficiently large), but takes much more development time.

7.2 Processing of data

The important requirement is that each algorithm is implemented in a way that accommodates processing of subsets of the larger data set. If that happens, then each of the "smart" techniques can be implemented using data flow networks and allowing the underlying system to control how data flows through. For in situ processing, the data flow network will be instantiated on the simulation code's processors and the data operated on will be each processor's portion of the data set. For out-of-core processing, one data flow network will be instantiated, and subsets of the larger data set will be read and sent through the network one at a time. For multi-resolution, the data at any given time will be some granularity of the larger data set, with previous executions of the network being wiped out as they get replaced by finer versions of the data set. For pure parallelism, each processor will get a data flow network and the data set is partitioned so that each processor gets a subset to work on.

Not all algorithms work smoothly in this setting. Algorithms that require all of the data to be in memory at one time cannot support out-of-core techniques, for example. However, one alternative is to have each algorithm communicate its needs with a data flow network "executive." The executive can then make decisions about which modes of operation are possible. The contracts mechanism proposed in [3] is ideal for this type of communication.

That said, resources for a given analysis are typically fixed while the analysis is running ("I will be running in-situ", "I have to run out-of-core", etc.), so the executive may be limited to aborting the operation and recommending a different processing paradigm.

7.3 Viability

Although the proposed system is daunting, the good news is that work has taken place to establish its viability. The VisIt project (of which the author of this paper is a member) targets all five of these use cases and has demonstrated promising results in the area of dynamic processing paradigms. VisIt has full support for pure parallelism. It also supports out-of-core processing, although some of its algorithms are not possible in an out-of-core environment. Those algorithms set a field in a contract and an executive disables the out-of-core mode. (It then falls back to pure parallelism.) VisIt also is capable of in-situ processing, although it suffers from the data model conversion issue described in section 3.1. There is no multi-resolution support, however[6]. Re-

[6]There is no support for level-of-detail processing. There is full support for Adaptive Mesh Refinement (AMR) data, but the user controls what level

gardless, the ability to provide an environment that allows for algorithms to be implemented indifferent of processing paradigms, combined with multiple processing paradigms being provided is promising.

8 SUMMARY

The ability to postprocess petascale data will be crucial to the success of petascale simulations. The current path being used for terascale postprocessing (pure parallelism) will result in prohibitive hardware costs. A software investment is needed to reduce these costs, in the form of a new system that can support a variety of "smart" processing techniques and employ them to diverse use cases.

REFERENCES

[1] Greg Abram and Lloyd A. Treinish. An extended data-flow architecture for data analysis and visualization. Research report RC 20001 (88338), IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, February 1995.

[2] Rajesh K. Batra and Lambertus Hesselink. Feature comparisons of 3-D vector fields using earth mover's dist ance. In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 105–114, San Francisco, 1999.

[3] Hank Childs, Eric Brugger, Kathleen Bonnell, Jeremy Meredith, Mark Miller, Brad Whitlock, and Nelson Max. A contract based system for large data visualization. In *Proceedings of IEEE Visualization 2005*, 2005.

[4] Hank Childs and Mark Miller. Beyond meat grinders: An analysis framework addressing the scale and complexity of large data sets (to appear). In *Proceedings of HPC2006*, 2006.

[5] Computational Engineering International, Inc. *EnSight User Manual*, May 2003.

[6] Willem C. de Leeuw, Hans-Georg Pagendarm, Frits H. Post, and Birgit Walter. Visual simulation of experimental oil-flow visualization by spot noise images from numerical flow simulation. In *Visualization in Scientific Computing '95*, pages 135–148, 1995.

[7] Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Local and global comparison of continuous functions. In *Proceedings of the IEEE conference on Visualization (VIS-04)*, pages 275–280, October 2004.

[8] C.R. Johnson, S. Parker, and D. Weinstein. Large-scale computational science applications using the SCIRun problem solving environment. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, 2000.

[9] C. Charles Law, Amy Henderson, and James Ahrens. An application architecture for large data visualization: a case study. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 125–128. IEEE Press, 2001.

of refinement is viewed, based on the available resources.

[10] Alain Mazzolo and Benoit Roesslinger. Monte-carlo simulation of the chord length distribution function across convex bodies, non-convex bodies and random media. *Journal of Mathematical Physics*, 44(12):6195–6208, 2003.

[11] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 93–ff. IEEE Computer Society Press, 1996.

[12] Qin Shen, Alex Pang, and Sam Uselton. Data level comparison of wind tunnel and computational fluid dynamics data. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 415–418, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[13] Qin Shen, Sam Uselton, and Alex Pang. Comparison of wind tunnel experiments and computational fluid dynamic s simulations. *Journal of Visualization*, 6(1):31–39, 2003.

[14] C. Silva, Y. Chiang, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. In *Visualization 2002 Course Notes*, 2002.

[15] O. Sommer and T. Ertl. Comparative visualization of instabilities in crash-worthiness simulat ions. In *Procceedings of EG/IEEE TCVG Symposium on Visualization VisSym '0 1*, pages 319–328, 2001.

[16] Craig Upson, Thomas Faulhaber Jr., David Kamins, David H. Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam. The application visualization system: A computational environment for scientific visualization. *Computer Graphics and Applications*, 9(4):30–42, July 1989.

[17] Peter L. Williams and Samual P. Uselton. Foundations for measuring volume rendering quality. Technical Report NAS/96-021, NASA Numerical Aerospace Simulation, 1996.

[18] Hualin Zhou, Min Chen, and Mike F. Webster. Comparative evaluation of visualization and experimental results using image comparison metrics. In *VIS '02: Proceedings of the conference on Visualization '02*, Washington, DC, USA, 2002. IEEE Computer Society.